ELEC-204 PRELIMINARY REPORT LAB-4

Name-Surname: Barış KAPLAN

ID Number: 69054

E-mail Address: bkaplan18@ku.edu.tr

The Date of Submission: 19 May 2021

1-)

PROVIDED LAB-4 TUTORIAL CODE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;


entity Lab4TutorialCode is
   Generic (N : INTEGER:=50*10**6); --50*10^6 Hz Clock
   Port ( MCLK : in  STD_LOGIC;-- Input 1 ,Type: STD_LOGIC, Name: MCLK
          X : in  STD_LOGIC;-- Input 2, Type: STD_LOGIC, Name: X
          Z : out  STD_LOGIC;-- Output 1 Type: STD_LOGIC, Name: Z
          CL : out  STD_LOGIC;-- Output 2, Type: STD_LOGIC, Name: CL
          CH : out  STD_LOGIC);-- Output 3, Type: STD_LOGIC, Name: CH
end Lab4TutorialCode;


architecture Behavioral of Lab4TutorialCode is
signal CLK_DIV: STD_LOGIC:='0';-- Intermediate signal 1, Type= STD_LOGIC, Name: CLK_DIV
```

```
signal State: STD_LOGIC:='0';--Intermediate signal 2, Type= STD_LOGIC, Name: State

begin

Z<=State xor X;

CH<=CLK_DIV;

CL<=not CLK_DIV;

process(MCLK)

variable Counter: INTEGER range 0 to N;-- Defining an integer variable called 'Counter'.

begin—beginning to the process called process(MCLK)

if(rising_edge(MCLK)) then// The rising_edge function returns a boolean value. Returns true if the value of the MCLK signal alters from 0 to 1.

Counter:=Counter+1;-- Incrementing the integer variable called 'Counter' by 1

if(Counter=N/1-1) then

Counter:=0;-- Setting the integer variable called 'Counter' to 0

CLK_DIV<=not CLK_DIV;

end if;-- Ending the if statement

end if;--Ending the if statement

end process;--Ending the process(MCLK)

process(CLK_DIV)

begin—beginning to the process called process(CLK_DIV)

if(rising_edge(CLK_DIV)) then

  State<=State xor X;

end if;-- Ending the if statement

end process;-- Ending the process(CLK_DIV)
```

LAB-4 TUTORIAL SIMULATION CODE(TEST BENCH PART)

```
LIBRARY ieee;

USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using

-- arithmetic functions with Signed or Unsigned values

--USE ieee.numeric_std.ALL;
```

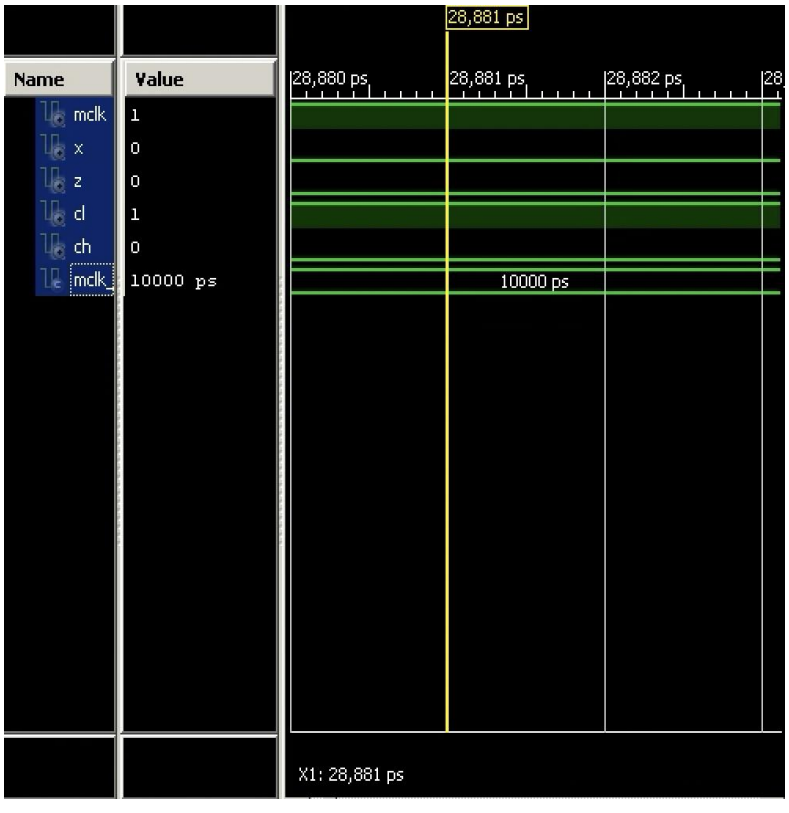
```
ENTITY Lab4TutorialSimulation IS
END Lab4TutorialSimulation;
ARCHITECTURE behavior OF Lab4TutorialSimulation IS
   COMPONENT Lab4TutorialCode
   PORT(
        MCLK : IN  std_logic; -- Input 1, Name: MCLK, Type: STD_LOGIC
        X : IN  std_logic; -- Input 2, Name: X, Type: STD_LOGIC
        Z : OUT  std_logic; --Output 1, Name: Z, Type: STD_LOGIC
        CL : OUT  std_logic;-- Output 2, Name: CL, Type: STD_LOGIC
        CH : OUT  std_logic - -Output 3, Name: CH, Type: STD_LOGIC
       );
   END COMPONENT;
  --Inputs
  signal MCLK : std_logic := '0';
  signal X : std_logic := '0';
            --Outputs
  signal Z : std_logic;
  signal CL : std_logic;
  signal CH : std_logic;
  -- Clock period definition with a constant value called 'MCLK_period'.
  constant MCLK_period : time := 10 ns;
BEGIN
            -- Instantiate the Unit Under Test (UUT)
  uut: Lab4TutorialCode PORT MAP (
        MCLK => MCLK,
        X => X,
        Z => Z,
        CL => CL,
        CH => CH
       );
  -- Stimulus process
  stim_proc: process
  begin
```

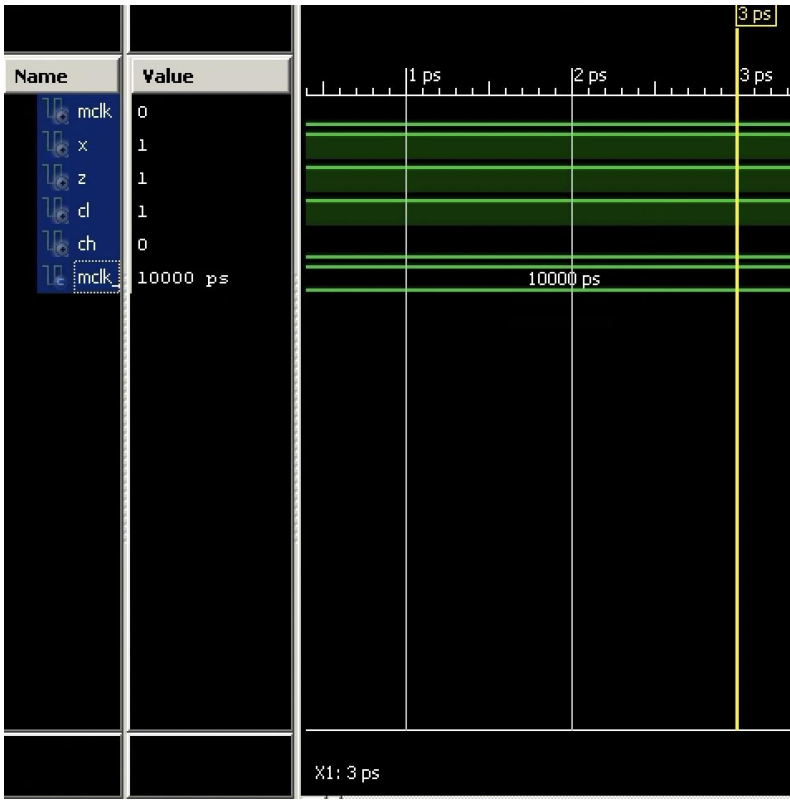
Changing the value of MCLK with a clock synchronization

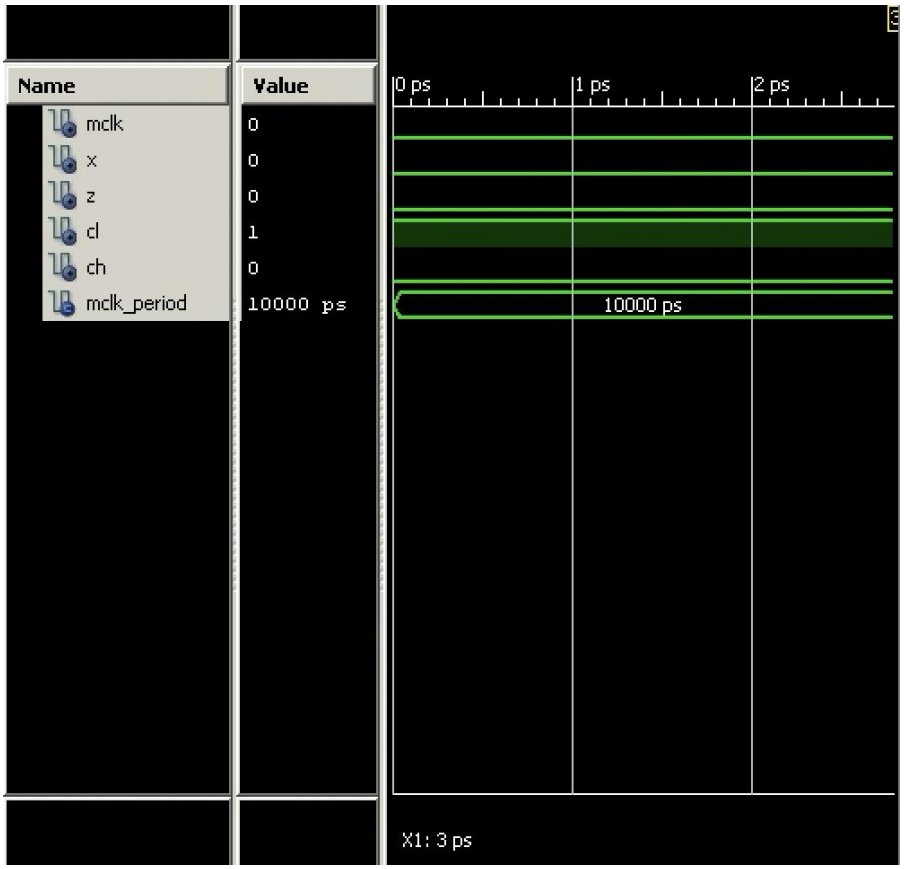
```
MCLK <= '0';    -- Here, X= 0 and MCLK=0
wait for MCLK_period/2; -- Wait until the time reaches MCLK_period/2
MCLK <= '1'; -- Here, X=1 and MCLK=1
wait for MCLK_period; -- Wait until the time reaches MCLK_period
MCLK <= '0'; --Here, X=1 and MCLK=0
wait for 3*MCLK_period/2; -- Wait until the time reaches 3*MCLK_period/2
MCLK <= '1';  -- Here, X=1 and MCLK= 1
-- insert stimulus here
wait;
end process;
        X <= '0', '1' after MCLK_period/2;-- Making X equal to 0 before the time MCLK_period /2, and --changing the value of X from 0 to 1 after the time MCLK_period/2
end behavior;
```

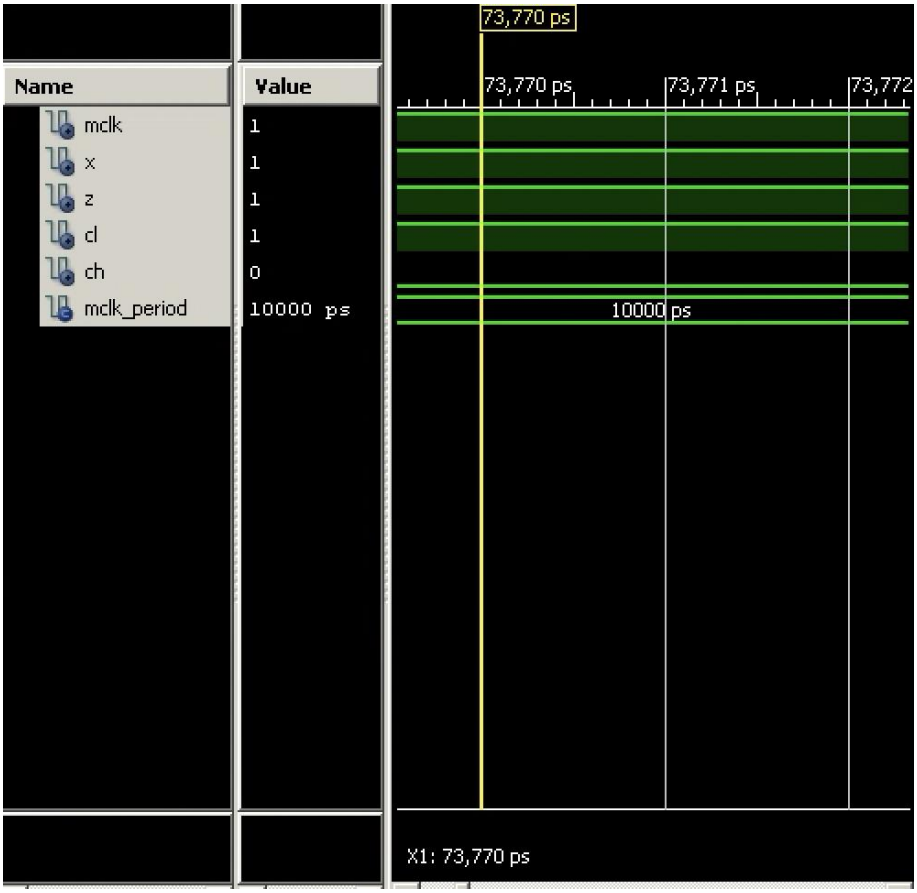
Note: We can observe MCLK= 1 and X= 0 case by swaping 0 and 1 in the statement that I have written before the 'end behavior' statement. If we do this, the value of the signal X will be equal to 0 after the MCLK_period/2. After 15 ns, the value of the MCLK will be equal to 1. So, we have obtained the MCLK= 1 and X= 0 case after we wait for 15 ns.

SCREENSHOTS FROM THE TIMING DIAGRAM(For CLK_DIV='0' and State='0')

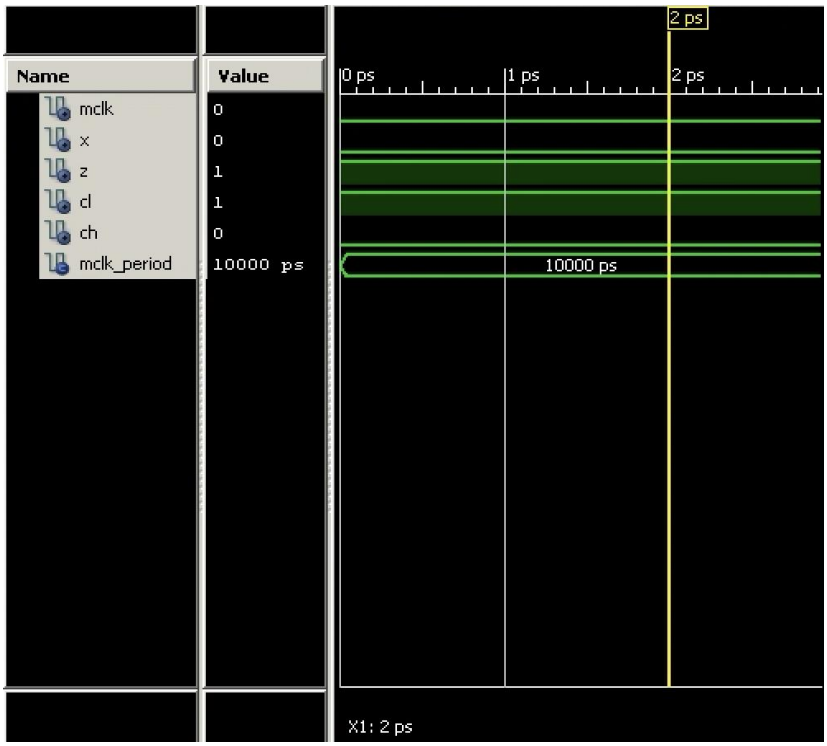


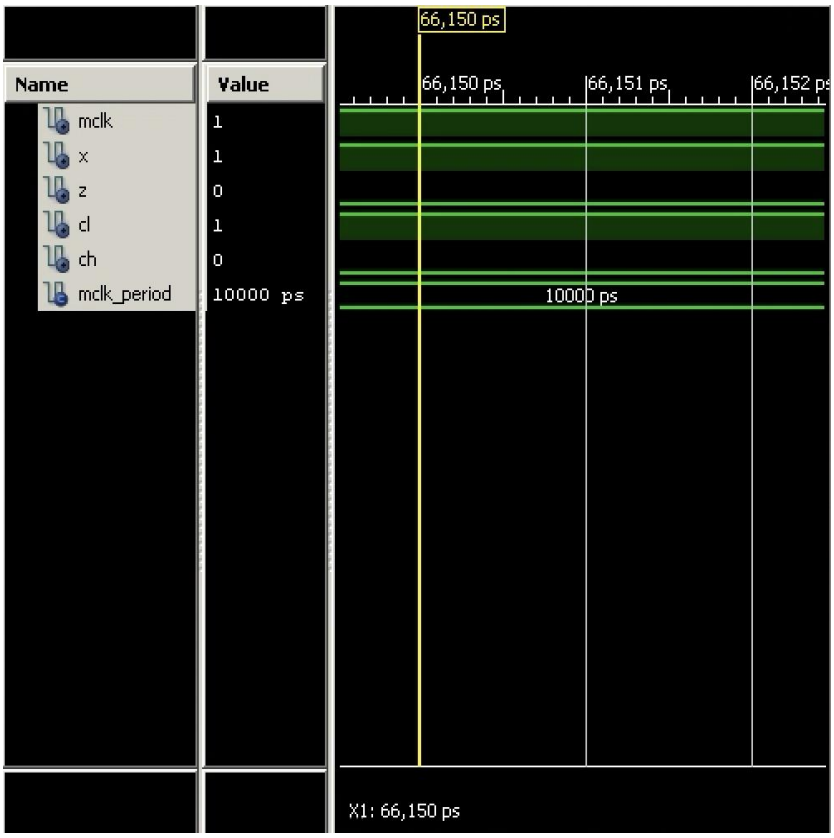


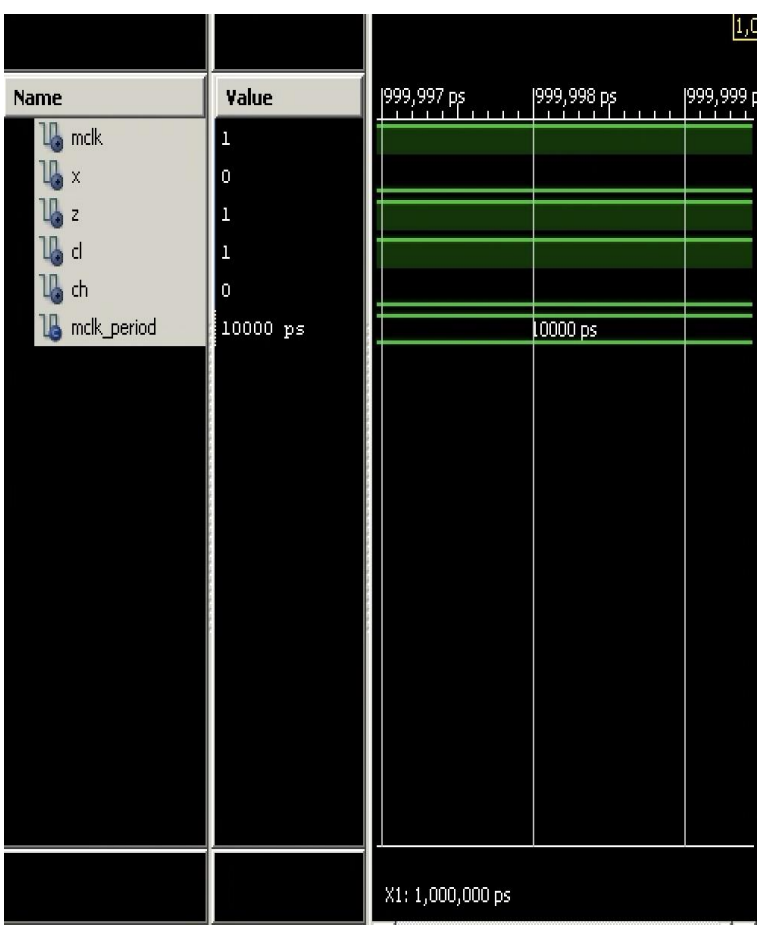

Name
Value
mclk
0
x
0
z
0
cl
1
ch
0
mclk_period
10000 ps
0 ps
1 ps
2 ps
10000 ps
X1: 3 ps

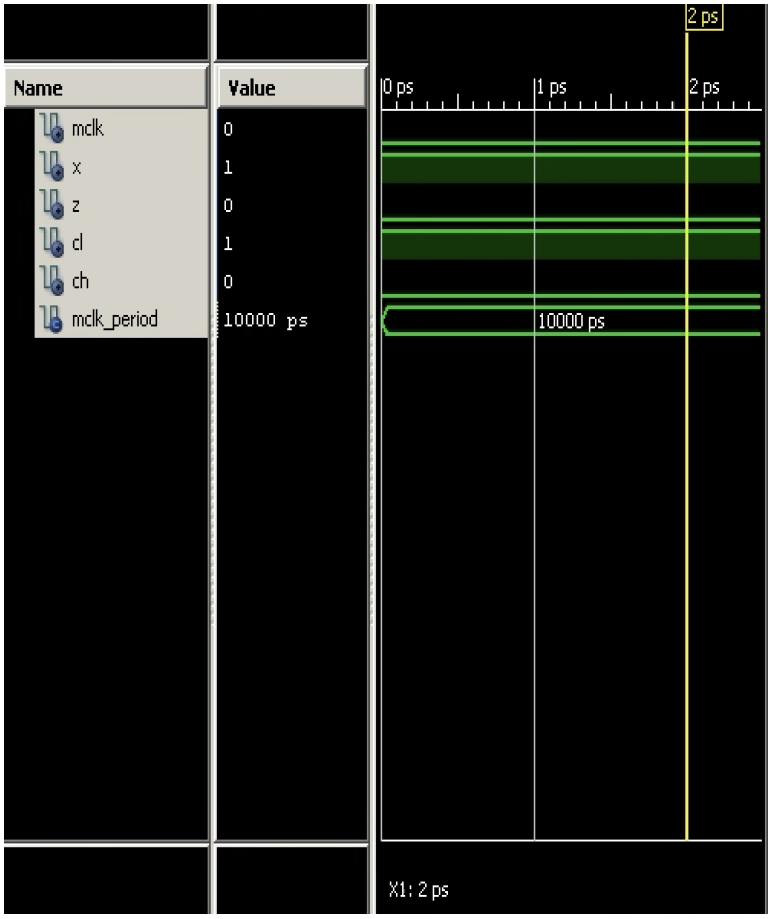

Name
Value
mclk
1
x
1
z
1
cl
1
ch
0
mclk_period
10000 ps
73,770 ps
73,770 ps
73,771 ps
73,772
10000 ps
X1: 73,770 ps

SCREENSHOTS FROM THE TIMING DIAGRAM(For CLK_DIV='0' and State='1')

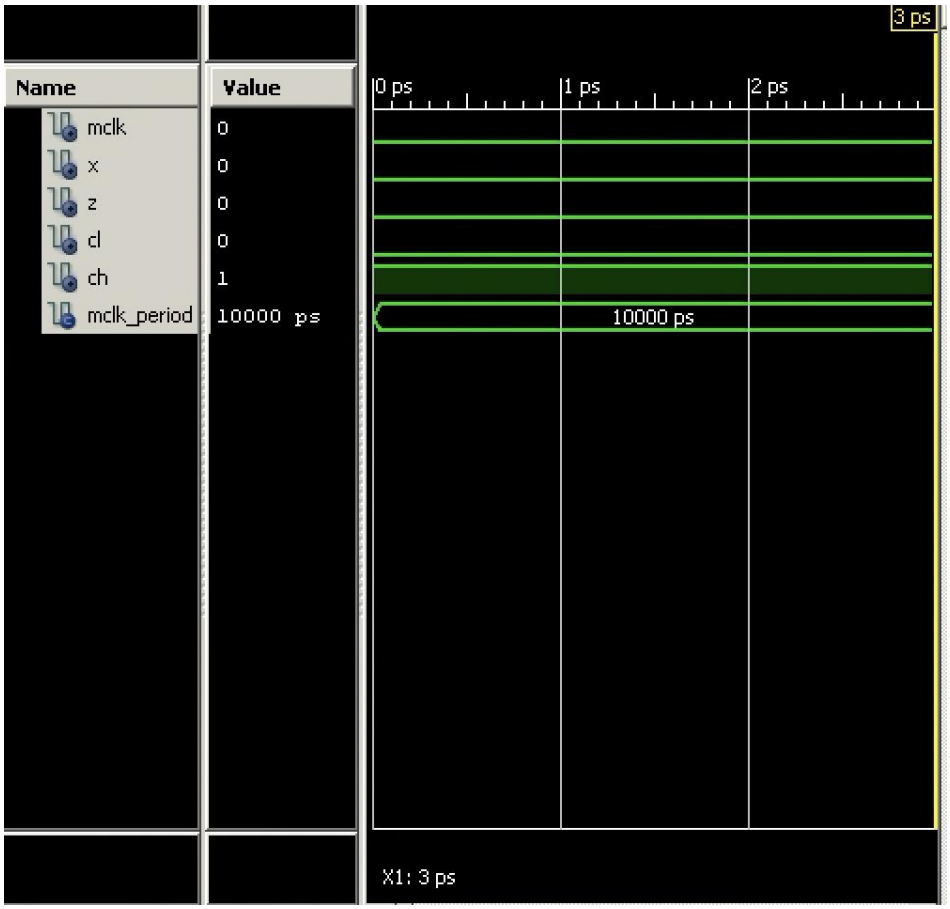


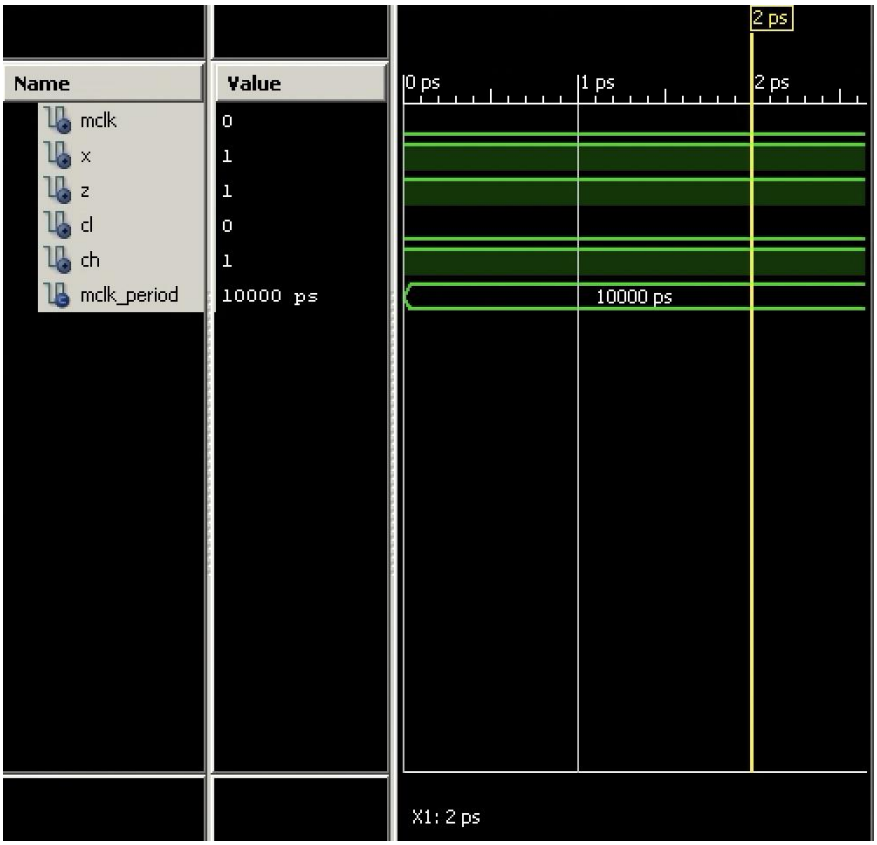


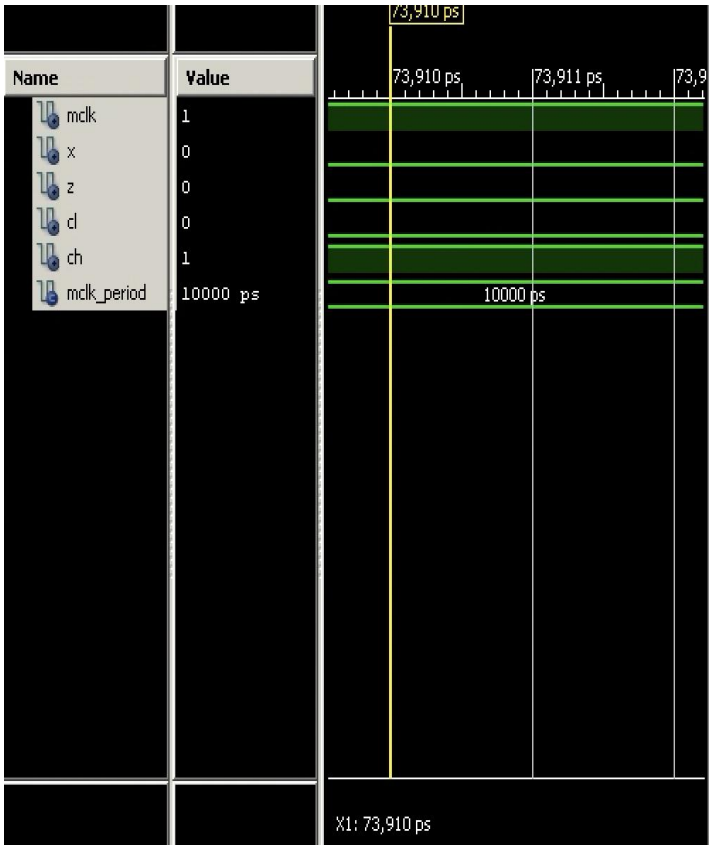


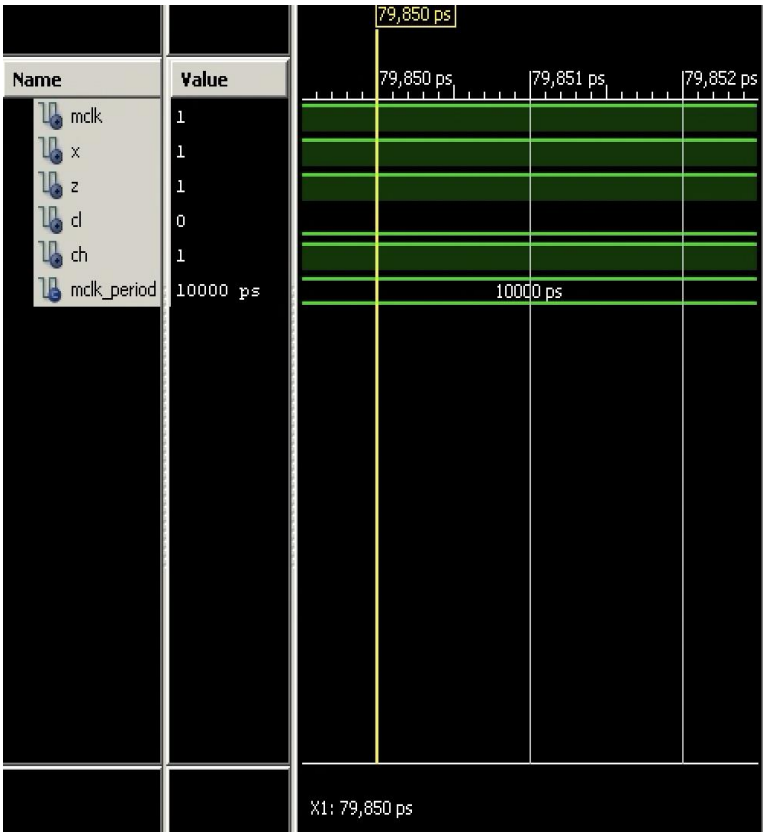


SCREENSHOTS FROM THE TIMING DIAGRAM (For CLK_DIV='1' and State='0')

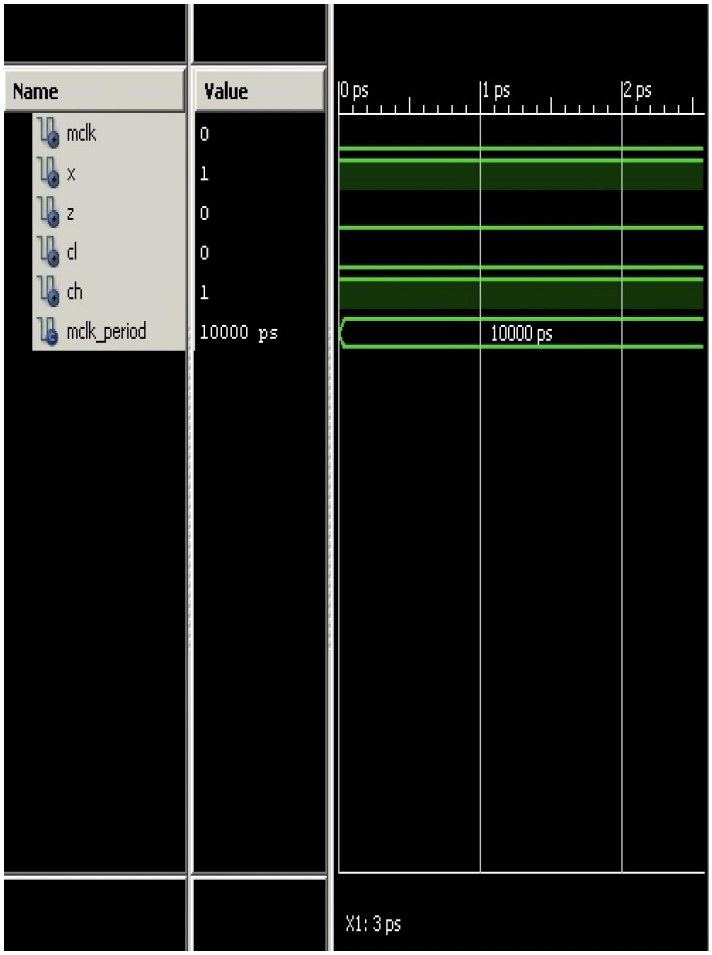


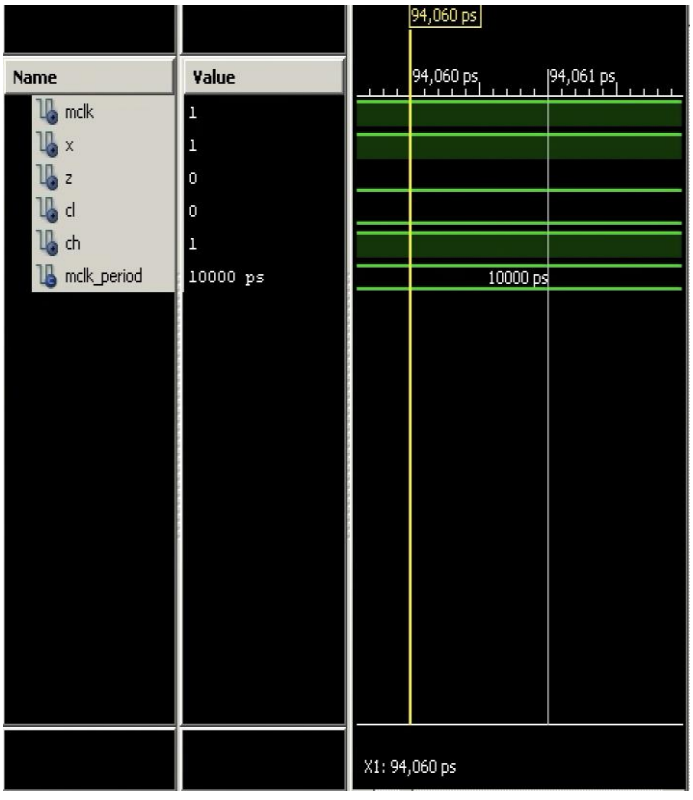

Name
Value
mclk
x
z
cl
ch
mclk_period
0
1
1
0
1
10000 ps
2 ps
0 ps
1 ps
2 ps
10000 ps
X1: 2 ps

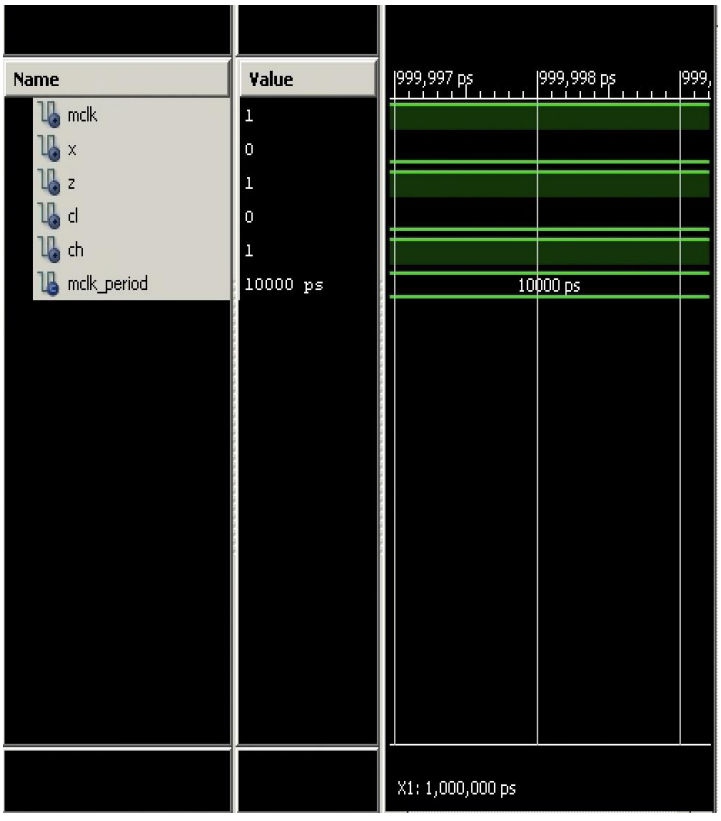

Name
Value
mclk
x
z
cl
ch
mclk_period
1
0
0
0
1
10000 ps
73,910 ps
73,910 ps
73,911 ps
10000 ps
X1: 73,910 ps


Name
Value
mclk
x
z
cl
ch
mclk_period
1
1
1
0
1
10000 ps
79,850 ps
79,850 ps
79,851 ps
79,852 ps
10000 ps
X1: 79,850 ps

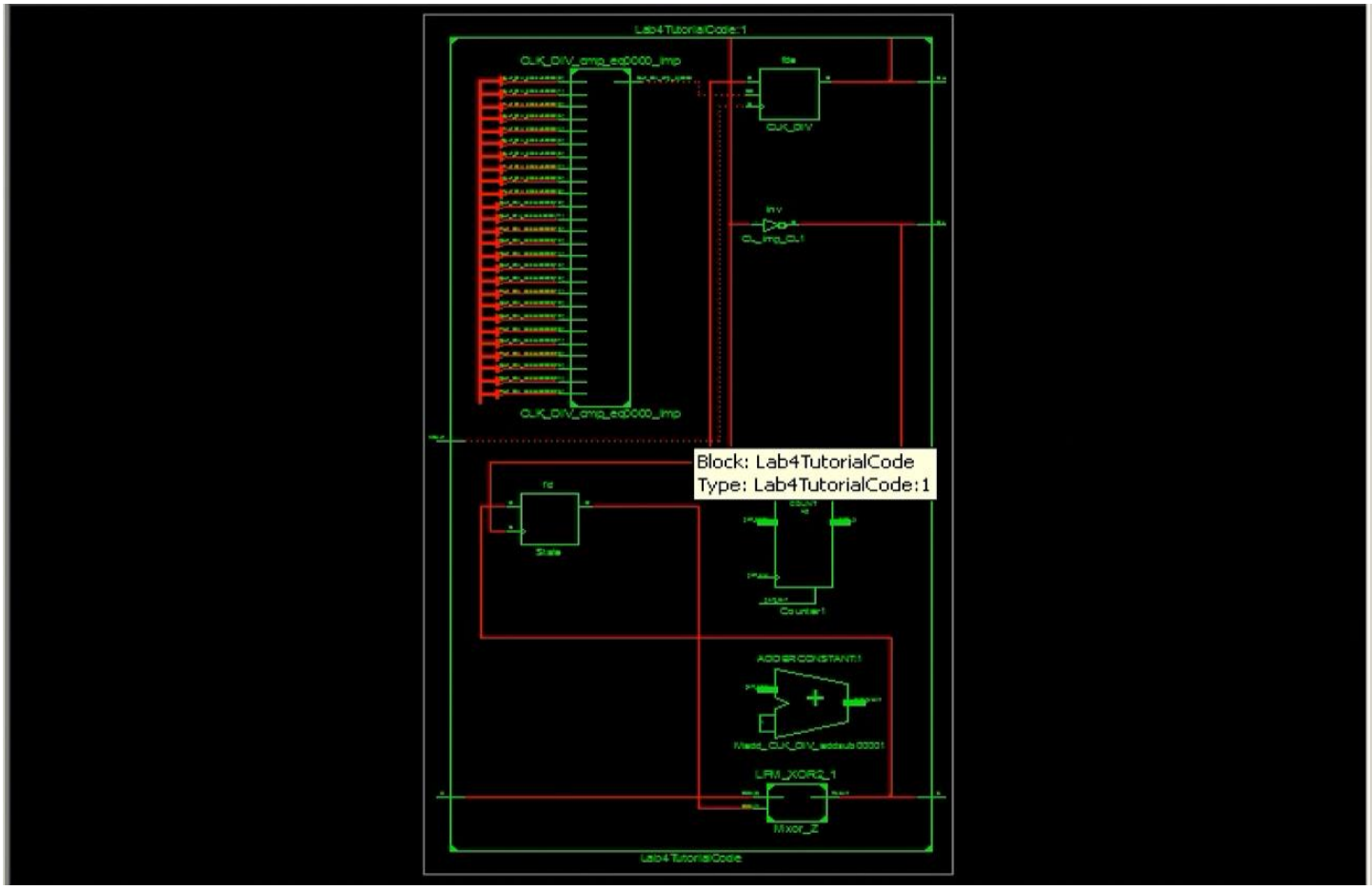
SCREENSHOTS FROM THE TIMING DIAGRAM(For CLK_DIV='1' and State='1')

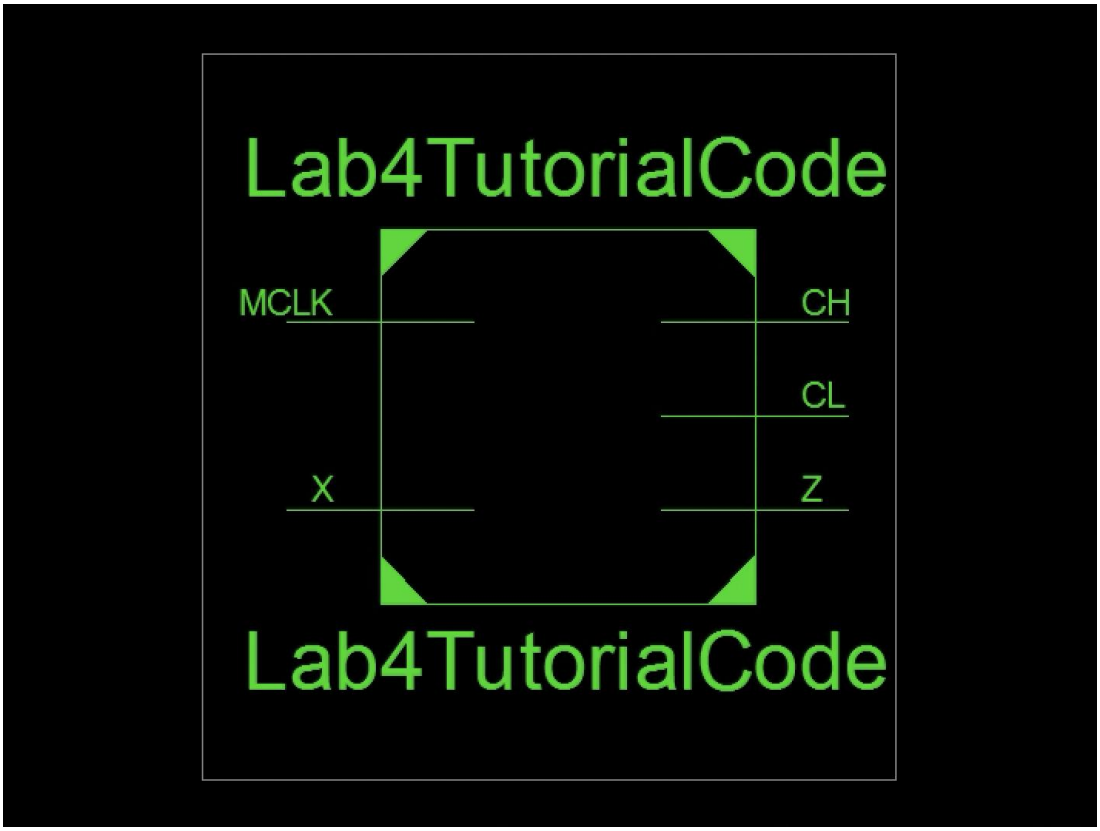






RTL SCHEMATIC OF THE PROVIDED LAB-4 TUTORIAL CODE





Explanation about the Inputs of the Provided Lab-4 Tutorial Code

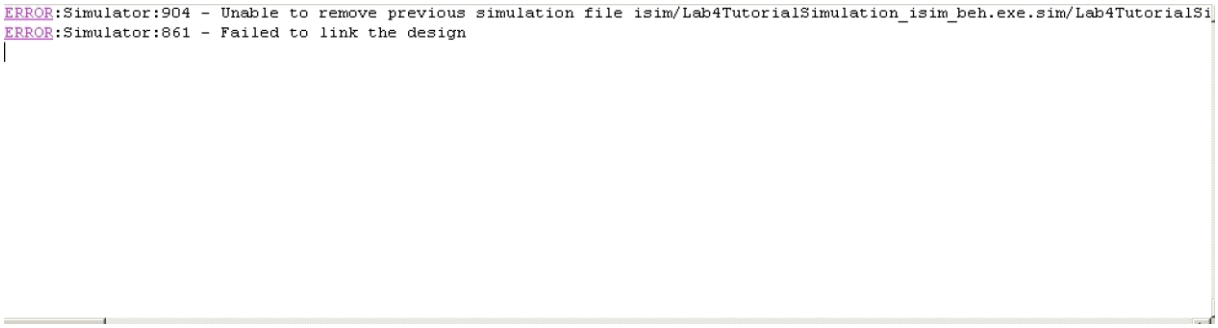
There are two inputs in total. The first input is called MCLK. The second input is called X. MCLK is a 1-bit input. X is a 1-bit input. X is the push button input. The type of the signal called MCLK is STD_LOGIC. The type of the signal called X is also STD_LOGIC.

Explanation about the outputs of the Provided Lab-4 Tutorial Code

There are three outputs in total. The first output is called Z. The second output is called CL. The third output is called CH. Z is the output which indicates whether the number of 1s in the clock synchronous input sequence is odd. If the number of 1s in the clock synchronous input sequence is odd, Z is equal to 1. If the number of 1s in the clock synchronous input sequence is even, Z is equal to 0. CL is the output for clock low. Moreover, CH is the output for clock high. Z is a 1-bit input. CL is a 1-bit input. CH is a 1-bit input. The type of the signal called Z is STD_LOGIC. The type of the signal called CL is STD_LOGIC. The type of the signal called CH is STD_LOGIC.

The errors that I have encountered &How I resolved this error/these errors

While I am simulating the provided lab-4 tutorial code, I have encountered with 1 error. The error is as follows:



NOTE: While I was studying the Lab-4 Tutorial Code and simulating this code, I did not use the FPGA board. So, I did not encountered with any FPGA board errors that I need to solve.

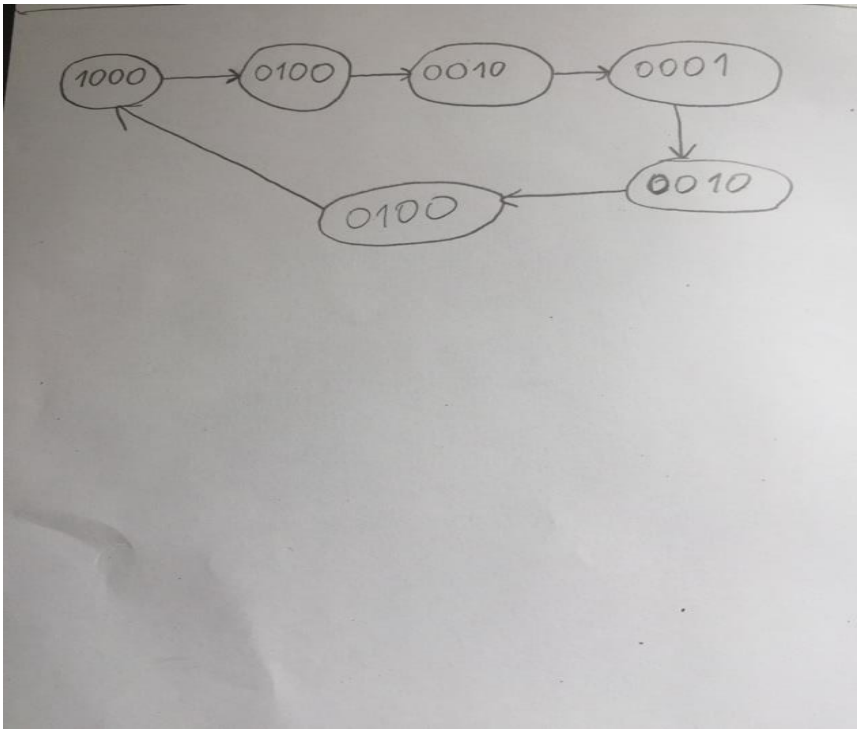
TRUTH TABLE OBTAINED FROM THE SIMULATION RESULTS

CLK_DIV	State	MCLK	X	Z	CL	CH
1	0	0	0	0	0	1
1	0	0	1	1	0	1
1	0	1	0	0	0	1
1	0	1	1	1	0	1
0	1	0	0	1	1	0
0	1	0	1	0	1	0
0	1	1	0	1	1	0
0	1	1	1	0	1	0
1	1	1	0	1	0	1
1	1	0	0	1	0	1
1	1	0	1	0	0	1

1	1	1	1	0	0	1
0	0	0	1	1	1	0
0	0	0	0	0	1	0
0	0	1	0	0	1	0
0	0	1	1	1	1	0

2-)

THE STATE DIAGRAM OF THE 4-BIT LED PATTERN



STATE TABLE OF THE 4-BIT LED PATTERN

Current State of the 4-Bit Led Display				Next State of the 4-Bit Led Display				D Flip-Flop Inputs				The direction bit	The next state of the direction bit
K	L	M	T	N_K	N_L	N_M	N_T	I3(I_K)	I2(I_L)	I1(I_M)	I0(I_T)	D	N_D
1	0	0	0	0	1	0	0	0	1	0	0	0	0
0	1	0	0	0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	0	0	1	0	0	0	1	0	1
0	0	0	1	0	0	1	0	0	0	1	0	1	1
0	0	1	0	0	1	0	0	0	1	0	0	1	1
0	1	0	0	1	0	0	0	1	0	0	0	1	0

N_D= K'L'MT'D' + K'L'M'TD + K'L'MT'D (Look at the row/rows where N_D is equal to 1)

FLIP-FLOP_INPUT EQUATIONS

I3(I_K)= K'LM'T'D (See the row/rows where I3(I_K) is equal to 1)
I2(I_L)= KL'M'T'D'+ K'L'MT'D (See the row/rows where I2(I_L) is equal to 1)
I1(I_M)= K'LM'T'D'+ K'L'M'TD (See the row/rows where I1(I_M) is equal to 1)
I0(I_T)= K'L'MT'D' (See the row/rows where I0(I_T) is equal to 1)

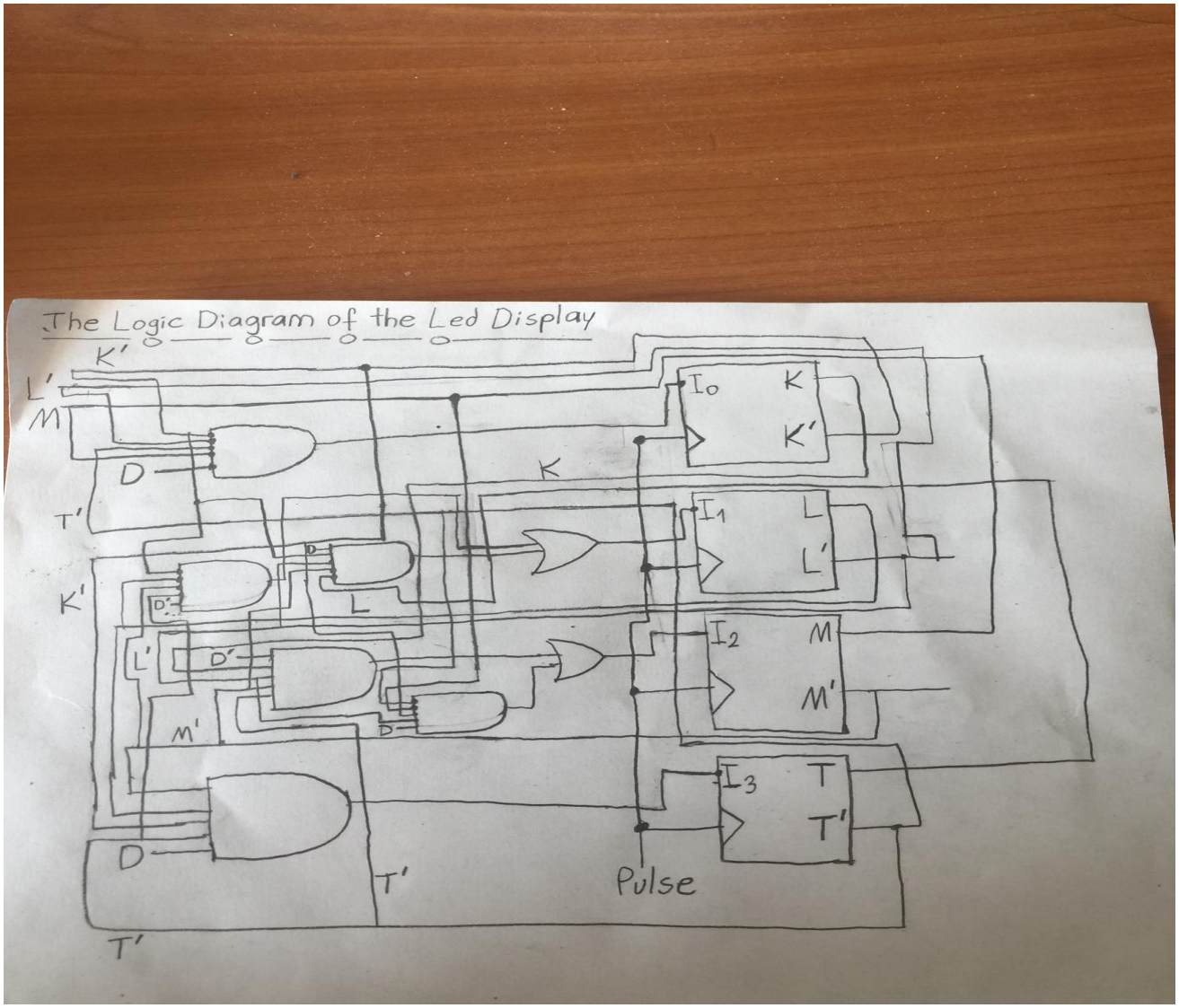
FLIP-FLOP OUTPUT EQUATIONS

N_K= I3(N_K=I_K)
N_L= I2(N_L=I_L)
N_M= I1(N_M=I_M)
N_T= I0(N_T=I_T)

EXPLANATIONS ABOUT THE DIRECTION BITS

->For the cases where KLMT= 1000, KLMT= 0100, and KLMT=0010; the direction bit is equal to 0(the bit which is equal to 1 is going from its' left-most position to its right-most position).

->For the cases where KLMT= 0001, KLMT= 0010, KLMT= 0100; the direction bit is equal to 1(the bit which is equal to 1 is going from its' right-most position to its left-most position).



Explanation about the led display design

While I am trying to construct the led display design, I have taken the flip-flop input equations into account. For the construction of the flip-flop input I3, I have used 1 AND gate with 5 inputs which are called K', L, M', T', and D. For the construction of the flip-flop input I2, I have used 2 AND gates with 5 inputs each (K,L',M',T', and D' for the first AND gate and K',L',M,T', and D for the second AND gate) and 1 OR gate with 2 inputs which are KL'M'T'D' and K'L'MT'D. For the construction of the flip-flop input I1, I have used 2 AND gates with 5 inputs each (K', L, M', T', and D' for the first AND gate and K', L', M', T, and D for the second AND gate) and 1 OR gate with 2 inputs which are K'LM'T'D' and K'L'M'TD. For the construction of the flip-flop input I0, I have used 1 AND gate with 5 inputs which are K', L', M, T', and D'.

3-)

SOME EXPLANATIONS ABOUT THE DESIGN OF THE BALL CATCHING INTERFACE

Let the first push button event be represented as Y1.
Let the second push button event be represented as Y2.

If the left user(Y1) presses the button when the LED pattern is equal to 1000, we should increment the score of the left-user by 1. If the right-user(Y2) presses the button when the LED pattern is equal to 0001, we should add 1 to the score of the right user.

THE STATE TABLE FOR THE DERIVATION OF THE BALL CATCHING INTERFACE

K	L	M	T	D	Y1	Y2	N_K	N_L	N_M	N_T	N_D	I3	I2	I1	I0
1	0	0	0	0	1(0)	0,1	0	1	0	0	0	0	1	0	0
0	1	0	0	0	0,1	0,1	0	0	1	0	0	0	0	1	0
0	0	1	0	0	0,1	0,1	0	0	0	1	1	0	0	0	1
0	0	0	1	1	0,1	1(0)	0	0	1	0	1	0	0	1	0
0	0	1	0	1	0,1	0,1	0	1	0	0	1	0	1	0	0
0	1	0	0	1	0,1	0,1	1	0	0	0	0	1	0	0	0

Here; K,L,M and T represent the present state variables.
Here; I3, I2, I1, and I0 represent the D flip-flop input variables.
Here; N_K, N_L, N_M, and N_T are the next state variables.
Here; D represents the direction bit.
Here; N_D represents the next state of the direction bit.
Here; Y1 and Y2 represent the push button events.

Note: We introduce 2 additional push button inputs which are called Y1 and Y2 in comparison to the state table of 4-bit led pattern.

FLIP-FLOP INPUT EQUATIONS

I3(I_K)= K'LM'T'D(Y1'Y2+Y2'Y1)
I2(I_L)= KL'M'T'D'(Y1'Y2+Y2'Y1) + K'L'MT'D(Y1'Y2+Y2'Y1)
I1(I_M)= K'LM'T'D'(Y1'Y2+Y2'Y1) + K'L'M'TD(Y1'Y2+Y2'Y1)

I0(I_T)= K'L'MT'D' (Y1'Y2+Y2'Y1)

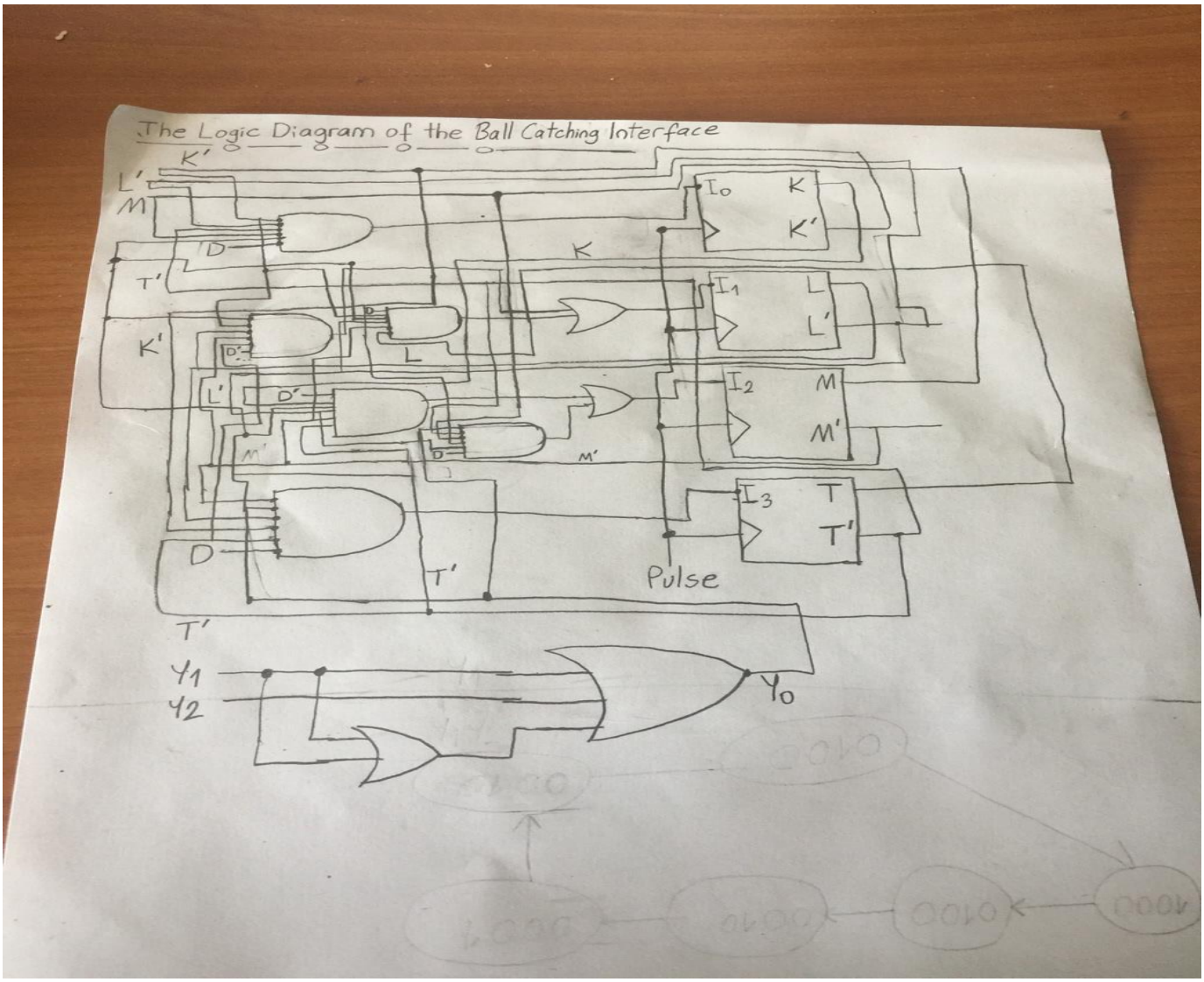
SOME EXPLANATIONS ABOUT THE STATE TABLE&FLIP FLOP EQUATIONS

In the state table, we can see that Y1 is exactly equal to 1 for the XKWZ= 1000 case. For the XKWZ= 0001 case, Y2 is exactly equal to 1. So, we should give Y1 as an input, and Y2 as the other input to an OR gate. Moreover, for the other cases, Y1 and Y2 can take any values. So, we should give also (Y1 or Y2) as an input to the OR gate. After that, to add the ball catching functionality, we should feed the result of this OR gate, which is Yo, into all of the AND gates in the circuit design of the ball catching interface.

OUTPUT FUNCTIONS

For the push button event part, the inputs to the OR gate are Y1, Y2, and (Y1 or Y2). This gives us the output equation as Y1 or Y2 or (Y1 or Y2), which is (Y1 or Y2) (1 or 1). This gives the simplified output equation as 'Y1 or Y2'.The next state outputs are equal to the flip-flop input bits. So, we can say that N_K = I3, N_L= I2, N_M= I1, and N_T = I0 are the next state output equations.

DESIGN OF THE BALL CATCHING INTERFACE



4-)

THE STATE TABLES OF THE 4-BIT COUNTER

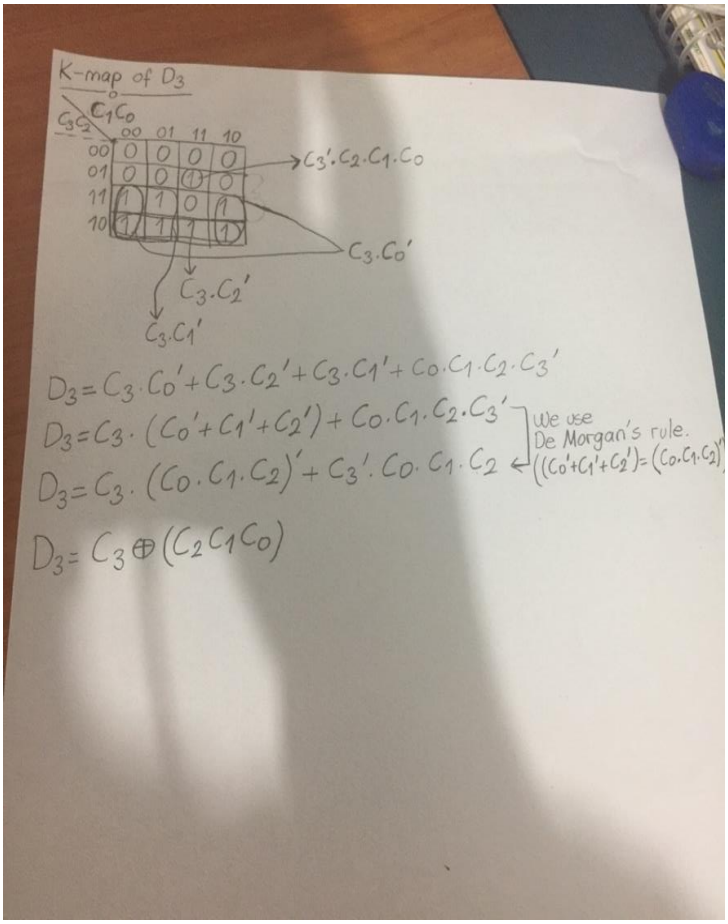
Current State of the value of 4-Bit Counter				Next State of the value of 4-Bit Counter				D Flip-Flop Inputs			
C3	C2	C1	C0	N3	N2	N1	N0	D3	D2	D1	D0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	0
0	0	1	0	0	0	1	1	0	0	1	1
0	0	1	1	0	1	0	0	0	1	0	0
0	1	0	0	0	1	0	1	0	1	0	1
0	1	0	1	0	1	1	0	0	1	1	0
0	1	1	0	0	1	1	1	0	1	1	1
0	1	1	1	1	0	0	0	1	0	0	0
1	0	0	0	1	0	0	1	1	0	0	1
1	0	0	1	1	0	1	0	1	0	1	0
1	0	1	0	1	0	1	1	1	0	1	1
1	0	1	1	1	1	0	0	1	1	0	0
1	1	0	0	1	1	0	1	1	1	0	1
1	1	0	1	1	1	1	0	1	1	1	0
1	1	1	0	1	1	1	1	1	1	1	1
1	1	1	1	0	0	0	0	0	0	0	0

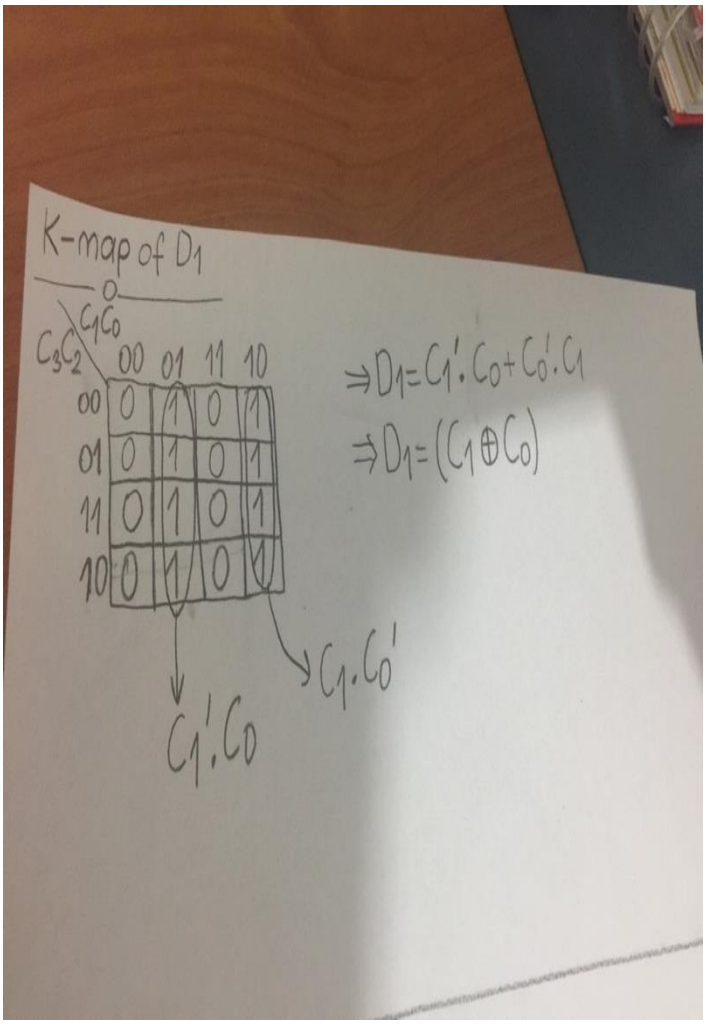
The current state of the value of 4-Bit Counter represents both the left user's and the right user's current total score. The next state of the value of 4-Bit Counter represents both the left user's and the right user's total score for the next state.

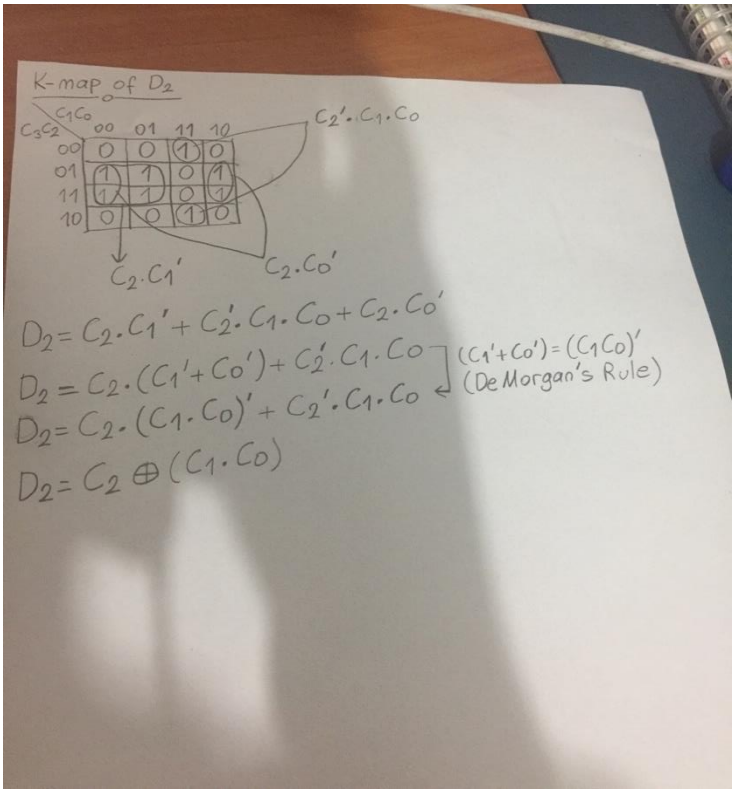
K	L	M	T	D	Y1	Y2	N_K	N_L	N_M	N_T	N_D	I3	I2	I1	I0	Incr_Y1	Incr_Y2
1	0	0	0	0	1(0)	0,1	0	1	0	0	0	0	1	0	0	1 (0)	0
0	1	0	0	0	0,1	0,1	0	0	1	0	0	0	0	1	0	0	0
0	0	1	0	0	0,1	0,1	0	0	0	1	1	0	0	0	1	0	0
0	0	0	1	1	0,1	1(0)	0	0	1	0	1	0	0	1	0	0	1 (0)
0	0	1	0	1	0,1	0,1	0	1	0	0	1	0	1	0	0	0	0
0	1	0	0	1	0,1	0,1	1	0	0	0	0	1	0	0	0	0	0

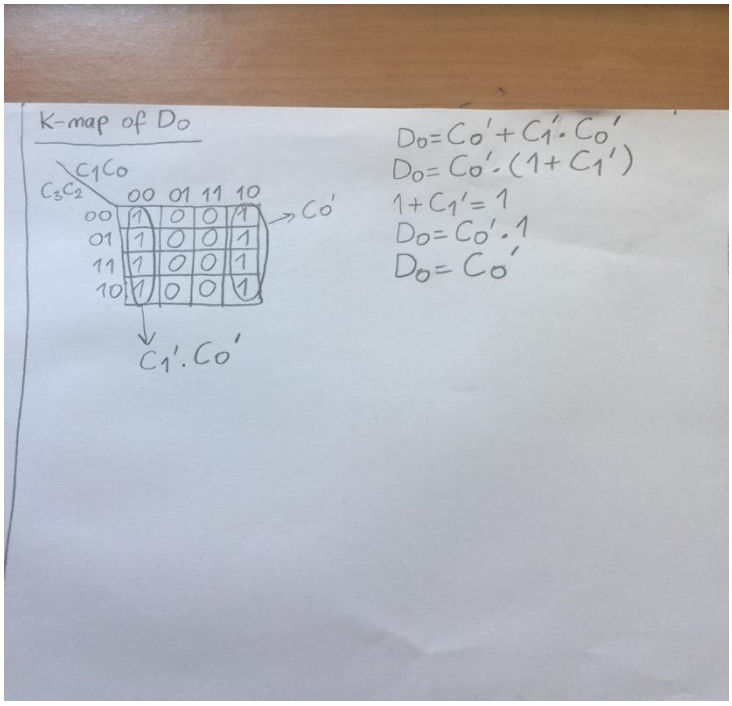
Here; K,L,M and T represent the present state variables for the 4-Bit led display.
Here; N_K, N_L, N_M, and N_T are the next state variables for the 4-Bit led display.
Here; D represents the direction bit.
Here; N_D represents the next state of the direction bit.
Here; Y1 and Y2 represent the push button events.
Here; I3, I2, I1, and I0 represent the D flip-flop input variables.
Here, Incr_Y1 is the bit which represents the incremention for the left user. The left user is represented as Y1.
Here, Incr_Y2 is the bit which represents the incremention for the right user. The right user is represented as Y2.

K-MAP'S OF THE D FLIP-FLOP INPUTS&OBTAINING THE INPUT EQUATIONS FROM KMAP









The rules that I used for deriving a boolean expression for D Flip-Flop Inputs

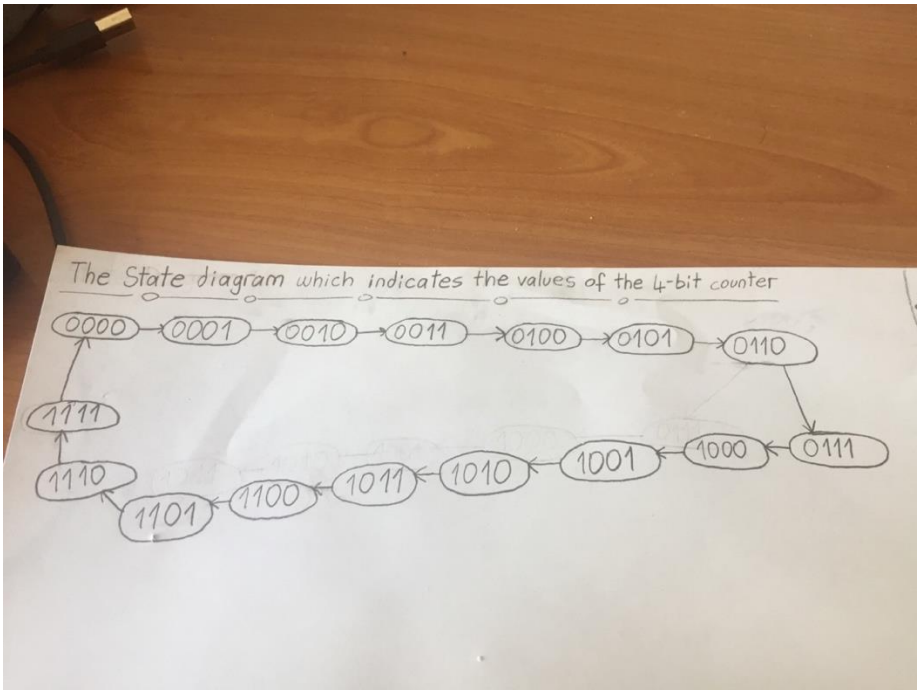
(x xor y)= (x' and y) or (y' and x)= (x'*y)+(y'*x)

(A+B)'=A'*B' (De Morgan's Rule)

A'+B'= (A*B)' (De Morgan's Rule)

1+C1'=1(One of the fundamental rules in boolean algebra)
(x' and y) or (y' and x) is the expanded form of (x xor y).

THE STATE DIAGRAM WHICH SHOWS THE VALUES OF THE 4-BIT COUNTER



MY CIRCUIT DESIGN OF THE 4-BIT COUNTER & EXPLANATIONS ABOUT MY CIRCUIT DESIGN OF 4-BIT COUNTER

- ➔ **I used D Flip-Flops in my 4-bit counter circuit design.**

To find a boolean expression for D0, I have drawn a K-Map of D0. While I am drawing the K-Map of D0, I have looked up to the values of C3,C2,C1, and C0 in the state table of the 4-bit counter and I have looked up whether the value of D0 is 1 or not. For an input/some inputs of C3 C2 C1 C0, if the value of D0 is 0; I have filled the corresponding places with 0 in the K-Map. Otherwise(D0 is 1); I have filled the corresponding places with 1 in the K-Map. After that, I have looked to the common terms of the groups in the K-Map. Finally, I have applied OR operation to these common terms and I have tried to simplify the expression by using some boolean algebra rules. K-Map part, I find that D0 is equal to C0'. As I did in the image of the circuit design of the 4-bit counter, I can directly connect C0' with D0 or I can use 1 NOT gate, give C0 as the input to this NOT gate, and connect the end of the NOT gate with D0.

To find a boolean expression for D1, I have drawn a K-Map of D1. While I am drawing the K-Map of D1, I have looked up to the values of C3,C2,C1, and C0 in the state table of the 4-bit counter and I have looked up whether the value of D0 is 1. For an input/some inputs of C3 C2 C1 C0, if the value of D1 is 0; I have filled the corresponding places with 0 in the K-Map. Otherwise(D1 is 1); I have filled the corresponding places with 1 in the K-Map. After that, I have looked to the common terms of the groups in the K-Map. At the end, I have applied OR operation to these common terms and I have tried to simplify the expression by using some boolean algebra rules.In the K-Map part, I find that D1 is equal to (C1 xor C0). We

should use 1 XOR gate, give C0 and C1 as the inputs of this XOR gate, and connect the end of the XOR gate with D1. We should use 1 XOR gate for this part of the circuit design.

To find a boolean expression for D2, I have drawn a K-Map of D2. While I am drawing the K-Map of D2, I have looked up to the values of C3,C2,C1, and C0 in the state table of the 4-bit counter and I have looked up whether the value of D2 is 1. For an input/some inputs of C3 C2 C1 C0, if the value of D2 is 0; I have filled the corresponding places with 0 in the K-Map. Otherwise(D2 is 1); I have filled the corresponding places with 1 in the K-Map. After that, I have looked to the common terms of the groups in the K-Map. Eventually, I have applied OR operation to these common terms and I have tried to simplify the expression by using some boolean algebra rules. In the K-Map part, I find that D2 is equal to (C2 xor (C1*C0)). Here, we should use 1 AND gate which has 2 inputs called C1 and C0 for doing the C1*C0 operation. Moreover, we should use 1 XOR gate for doing the C2 xor (C1*C0) operation. In total, we need 2 logic gates here.

To find a boolean expression for D0, I have drawn a K-Map of D3. While I am drawing the K-Map of D3, I have looked up to the values of C3,C2,C1, and C0 in the state table of the 4-bit counter and I have looked up whether the value of D3 is 1. For an input/some inputs of C3 C2 C1 C0, if the value of D3 is 0; I have filled the corresponding places with 0 in the K-Map. Otherwise(D3 is 1); I have filled the corresponding places with 1 in the K-Map. After that, I have looked to the common terms of the groups in the K-Map. Ultimately, I have applied OR operation to these common terms and I have tried to simplify the expression by using some boolean algebra rules.In the K-Map part, I find that D3 is equal to (C3 xor (C2*C1*C0)). Here, we should use 1 AND gate which has 3 inputs called C2, C1 and C0 for doing the C2*C1*C0 operation. In addition to that, we should use 1 XOR gate for doing the C3 xor (C2*C1*C0) operation. In total, we need 2 logic gates here.